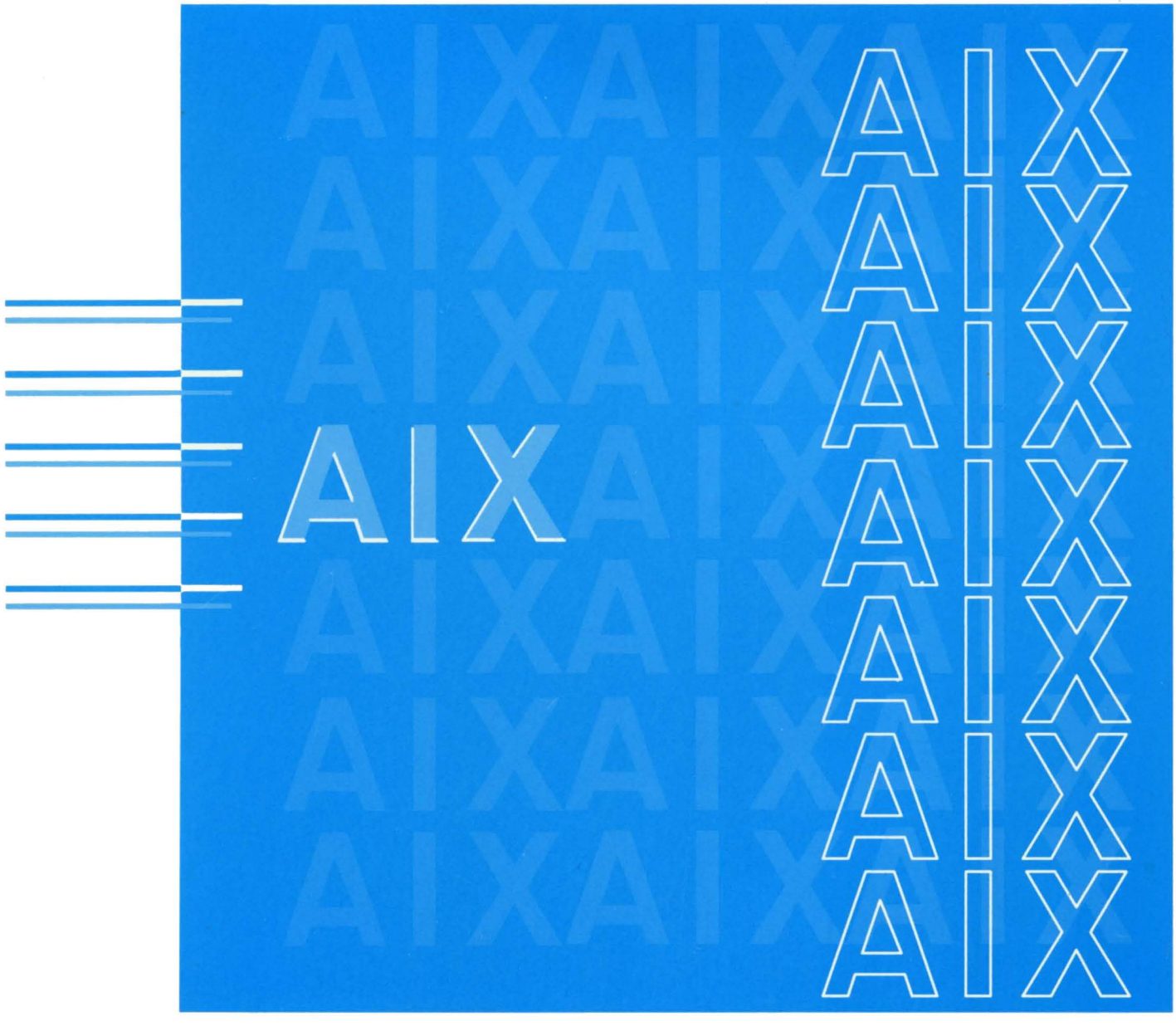
PCILIB,AIX Access for DOS Users

Extended Library



IBM

PCILIB
AIX Access for DOS Users Extended Library

March 1991

SC23–2360

First Edition (March 1991)

This edition applies to Version 2.1 of AIX Access for DOS Users, Program Number 5765-024, and to all subsequent releases until otherwise indicated in new editions or technical newsletters. Changes are made periodically to the information herein; these changes will be reported in technical newsletters or in new editions of this publication.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

International Business Machines Corporation provides this manual "as is," without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this manual at any time.

Publications are not stocked at the address given below. Requests for copies of this product and for technical information about the system should be made to your authorized IBM dealer or your IBM marketing representative.

A reader's comment form is provided at the back of this publication. If the form has been removed, address comments to IBM Corporation, Information Development, Department 52QA MS 911, Neighborhood Road, Kingston, NY 12401-0000. IBM may use or distribute, in any way it believes appropriate and without incurring any obligation to the sender, whatever information it receives in this manner.

© Copyright International Business Machines Corporation 1990
© Copyright Locus Computing Corporation 1988, 1989, 1990

Note to US Government Users—Documentation related to restricted rights—Use, duplication, or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Trademarks

The following trademarks apply to this book:

- AIX is a trademark of International Business Machines Corporation.
- Excelan is a registered trademark of Excelan, Inc.
- EXOS 205 is a trademark of Excelan, Inc.
- Ethernet is a trademark of Xerox Corporation.
- IBM, PC-DOS, PC XT, PC AT, and PS/2 are trademarks of International Business Machines Corporation.
- EtherLink and 3Com are trademarks of 3Com Corporation.
- PC-Interface and Merge are trademarks of Locus Computing Corporation.
- VT100 and VT220 are trademarks of Digital Equipment Corporation.
- XEROX is a registered trademark of Xerox Corporation.

Contents

Introduction 3

What is PCILIB? 4

PCILIB Distribution Diskette 5
- Library Formats 5
- Memory Models Supported 6
- Include Files 6
- Manifest Constants 7

intro 9
lcs_intro 15
dflthost 18
feature 20
getuattr 23
isvirtual 24
lcs_get_table 25
lcs_release_table 26
lcs_set_options 27
lcs_set_tables 29
lcs_translate_block 30
lcs_translate_string 32
mapd2u 34
mapu2d 36
msgctl 38
msgget 40
msgrcv 42
msgsnd 45
semctl 47
semget 50
semop 52
sleep_h 55
uchmod 57
uexec 59
ukill 62
uren 65
uwait 66
vdrive 68
vhost 70

contents

Introduction

PCILIB is a programming library that allows DOS programs written in C or assembly language to access the extended functions of AIX Access for DOS Users.

Developers have found the functions provided internally in the Access program to be very useful for implementing sophisticated DOS applications on top of the Access program virtual file service. Such applications include office automation systems, integrated DOS and AIX operating system mail handlers, and direct communication systems operating between distributed DOS and AIX operating system applications.

PCILIB provides the DOS programmer with functions to do the following:

- Determine and change the attributes of files on virtual drives.
- Find the virtual drive associated with a given DOS path.
- Map DOS file names to AIX operating system file names.
- Map AIX operating system file names to DOS file names.
- Find the drive numbers of virtual drives.
- Find the names of connected hosts.
- Execute AIX operating system processes remotely from the DOS environment.
- Return information regarding the currently supported features.
- Provide interprocess communication between personal computers and a host via AIX–compatible message queues and semaphores.
- Translate from one code page or code set to another so that a viewable character (glyph) under one code page or code set is also recognizable under another code page or code set.

This document assumes that you are familiar with DOS, the Microsoft C programming environment, and the Access program. It also assumes that you have some familiarity with AIX operating systems. Users who need more information on any of these topics should refer to the following documents:

- *AIX Access for DOS Users Administrator's Guide.* The chapter called The Access Program System Architecture provides details on how the Access program works.
- *Microsoft C Compiler User's Guide.* This document describes the C programming environment offered by the Microsoft Corporation.

- *AIX Operating System Technical Reference Volumes 1 and 2.* These technical reference manuals describe the programming interface and provide reference information on subroutines, system calls, file formats, and special files.

What Is PCILIB?

PCILIB is a library of DOS object modules, in Microsoft C format, that can be linked with user-written DOS programs to take advantage of the advanced features of the Access program virtual file service and System V–compatible interprocess communications (IPC) facilities.

The following table summarizes the extended functions provided by PCILIB:

DFLTHOST	Gets and sets the host to be used for subsequent System V–compatible IPC operations.
FEATURE	Gets features supported by a virtual drive.
GETUATTR	Gets host file attributes on a virtual drive.
ISVIRTUAL	Returns the virtual drive number of a specified path.
MAPD2U	Maps a DOS path name to an AIX operating system path name.
MAPU2D	Maps an AIX operating system path name to a DOS path name.
MSGCTL	Provides System V–compatible message control operations.
MSGGET	Provides System V–compatible "get message queue" operation.
MSGRCV	Provides System V–compatible "receive message" operation.
MSGSND	Provides System V–compatible "send message" operation.
SEMCTL	Provides System V–compatible control operations on network semaphores.
SEMGET	Provides System V–compatible operation for obtaining network semaphores.
SEMOP	Provides System V–compatible semaphore operations.
SLEEP_H	Provides the ability to give up control of the processor for a specified time.
UCHMOD	Changes file attributes on the virtual drive.
UEXEC	Executes an AIX operating system program on a host.

UKILL	Sends a signal to an AIX operating system task.
UREN	Renames an AIX operating system file.
UWAIT	Polls the status of an AIX operating system task initiated with UEXEC.
VDRIVE	Gets the virtual drive number of an Access program volume.
VHOST	Gets the name of a connected host.

Any AIX host running the Access program, Version 2.1 or later, with the IPC system enabled in the kernel, can honor all of the above PCILIB requests from DOS programs linked with the library. If the IPC system is not enabled, the message queue and semaphore operations are not available.

The following table summarizes the language character set (LCS) functions provided by PCILIB:

LCS_GET_TABLE	Gets the language character set conversion table.
LCS_RELEASE_TABLE	Releases a language character set conversion table.
LCS_SET_OPTIONS	Sets the language character set conversion table options.
LCS_SET_TABLES	Sets the input and output language character set conversion tables.
LCS_TRANSLATE_BLOCK	Translates the input block to the output block.
LCS_TRANSLATE_STRING	Translates the input string to the output string.

PCILIB Distribution Diskette

PCILIB is distributed on a 5¼–inch floppy diskette or 3½–inch diskette.

Library Formats

The PCILIB diskette contains a set of Microsoft library modules.

Memory Models Supported

PCILIB supports all memory models for the Microsoft C compiler, versions 5.0 and 5.1. Each memory model is in a separate subdirectory. Within each subdirectory, a file called PCIOEM.LIB contains the subdirectory contents. The following table lists the directories:

MSCC Compact memory model for Microsoft C.

MSCS Small memory model for Microsoft C.

MSCM Medium memory model for Microsoft C.

MSCL Large and Huge memory models for Microsoft C.

The default memory model is the "Small memory" model. Be sure to link the application program with the correct memory model library to ensure proper operation.

Include Files

The PCILIB distribution diskette also contains the directory INCLUDE. This directory comprises three include files for C programs that make calls to the PCILIB functions. The contents of this directory are as follows:

PCILIB.H Defines the external PCILIB function–call return types. This file should always be included in programs that use PCILIB function calls.

IPC.H Contains defines and types that are used in IPC message queues and semaphore operations. This file should be included in programs making IPC calls.

MEMMDL.H Contains the manifest constants for compiling with the Microsoft C compiler. This file must *always* be included in programs that make PCILIB-supported calls.

LCS.H Contains defines and global variables used for language character set translations. This file should always be included in programs making language character set calls.

Some of the PCILIB functions set the ERRNO variable when they return an error condition. In order to examine the DOS ERRNO variable, the calling program should include the system include file ERRNO.H. Consult your C compiler user's guide to determine where to find this file and how to make use of the variable.

Manifest Constants

You must use the MICROSOFT manifest constant when you compile C programs that make PCILIB calls. You should use the /D command-line option of the compiler, for example:

```
cl /DMICROSOFT EXAMPLE.C /link \pcilib\mscs\pcioem
```

NAME

intro – introduction to message queues and semaphores

SYNOPSIS

This section describes message queues, semaphore permissions, and data structures.

DESCRIPTION

Message Queue Identifier

A message queue identifier (**msqid**) is a unique positive integer created by a **msgget** system call. Each **msqid** has a message queue and a data structure associated with it. The data structure is referred to as **msqid_ds** and contains the following members:

```
struct ipc_perm msg_perm; /* operation permission struct */
time_t msg_stime;         /* last msgsnd time */
time_t msg_rtime;         /* last msgrcv time */
time_t msg_ctime;         /* last change time */
                          /* Times measured in secs since */
                          /* 00:00:00 GMT, Jan. 1, 1970 */
ushort msg_cbytes;        /* current number of bytes on q */
ushort msg_qnumn;         /* number of msgs on q */
ushort msg_qbytes;        /* max number of bytes on q */
ushort msg_lspid;         /* pid of last msgsnd operation */
ushort msg_lrpid;         /* pid of last msgrcv operation */
```

msg_perm is an **ipc_perm** structure that specifies the message operation permission (see below). This structure includes the following members:

```
ushort  uid;     /* user ID */
ushort  gid;     /* group ID */
ushort  cuid;    /* creator user ID */
ushort  cgid;    /* creator group ID */
ushort  mode;    /* r/w permission */
ushort  seq;     /* slot usage sequence number */
key_t   key;     /* key */
```

msg_stime is the time of the last **msgsnd** operation. **msg_rtime** is the time of the last **msgrcv** operation. **msg_ctime** is the time of the last **msgctl** operation that changed a member of the above structure. **msg_cbytes** is the number of bytes currently on the queue. **msg_qnum** is the number of messages currently on the queue. **msg_qbytes** is the maximum number of bytes allowed on the queue. **msg_lspid** is the process ID of the last process that performed a **msgsnd** operation. **msg_lrpid** is the process ID of the last process that performed a **msgrcv** operation.

Message Operation Permissions

In the **msgctl**, **msgrcv**, and **msgsnd** system call descriptions, the permission required for an operation is given as "{*token*}", where *token* is the type of permission needed, interpreted as follows:

00400	Read by user
00200	Write by user
00040	Read by group
00020	Write by group
00004	Read by others
00002	Write by others

Following usual AIX operating system permission conventions, permissions can be combined. For example, 00600 means read and write by user; 00640 means read and write by user and read by group.

Read and write permissions on an **msqid** are granted to a process if one or more of the following are true:

- The effective user ID of the process is super-user.
- The effective user ID of the process matches **msg_perm.[c]uid** in the data structure associated with **msqid**, *and* the appropriate bit of the "user" portion (0600) of **msg_perm.mode** is set.
- The effective user ID of the process does not match **msg_perm.[c]uid**, *and* the effective group ID of the process matches **msg_perm.[c]gid**, *and* the appropriate bit of the "group" portion (060) of **msg_perm.mode** is set.
- The effective user ID of the process does not match **msg_perm.[c]uid**, *and* the effective group ID of the process does not match **msg_perm.[c]gid**, *and* the appropriate bit of the "other" portion (06) of **msg_perm.mode** is set.

Otherwise, the corresponding permissions are denied.

Semaphore Identifier

A semaphore identifier (**semid**) is a unique positive integer created by a **semget** system call. Each semid has a set of semaphores and a data structure associated with it. The data structure is referred to as **semid_ds** and contains the following members:

```
struct ipc_perm sem_perm; /* operation permission struct */
time_t sem_otime;         /* last operation time */
time_t sem_ctime;         /* last change time */
                          /* Times measured in secs since */
                          /* 00:00:00 GMT, Jan. 1, 1970 */
ushort sem_nsems;         /* number of sems in set */
```

sem_perm is an **ipc_perm** structure that specifies the semaphore operation permission (see above).

sem_nsems is equal to the number of semaphores in the set. Each semaphore in the set is referenced by a positive integer referred to as a **sem_num**. **Sem_num** values run sequentially from 0 to the value of **sem_nsems** minus 1. **sem_otime** is the time of the last **semop** operation. **sem_ctime** is the time of the last **semctl** operation that changed a member of the above structure.

A semaphore is a data structure that contains the following members:

```
ushort  semval;   /* semaphore value */
short   sempid;   /* pid of last operation  */
ushort  semncnt;  /* # awaiting semval > cval */
ushort  semzcnt;  /* # awaiting semval = 0 */
```

semval is a nonnegative integer. **sempid** is equal to the process ID of the last process that performed a semaphore operation on this semaphore. **semncnt** is a count of the number of processes that are currently suspended awaiting this semaphore's **semval** to become greater than its current value. **semzcnt** is a count of the number of processes that are currently suspended awaiting this semaphore's **semval** to become zero.

Semaphore Operation Permissions

In the **semop** and **semctl** system call descriptions, the permission required for an operation is given as "{*token*}", where *token* is the type of permission needed, interpreted as follows:

00400	Read by user
00200	Alter by user
00040	Read by group
00020	Alter by group
00004	Read by others
00002	Alter by others

Following usual AIX operating system permission conventions, permissions can be combined. For example, 00600 means read and alter by user; 00640 means read and alter by user and read by group.

Read and alter permissions on a **semid** are granted to a process if one or more of the following are true:

- The effective user ID of the process is super-user.
- The effective user ID of the process matches **sem_perm.[c]uid** in the data structure associated with **semid**, *and* the appropriate bit of the "user" portion (0600) of **sem_perm.mode** is set.
- The effective user ID of the process does not match **sem_perm.[c]uid**, *and* the effective group ID of the process matches **sem_perm.[c]gid**, *and* the appropriate bit of the "group" portion (060) of **sem_perm.mode** is set.
- The effective user ID of the process does not match **sem_perm.[c]uid**, *and* the effective group ID of the process does not match **sem_perm.[c]gid**, *and* the appropriate bit of the "other" portion (06) of **sem_perm.mode** is set.

Otherwise, the corresponding permissions are denied.

USAGE NOTES

Using Correct Argument Types

It is important to use arguments of the correct type when writing an application using PCILIB. In many cases, the compiler cannot detect argument type mismatches; if an argument of the wrong type is used, the application may fail in an unpredictable way. The best way to be sure the arguments are of the correct type is to make appropriate type declarations or use appropriate type casts.

An example is the **key** argument to the **msgget** call. The argument to this function is of type **key_t**, which is defined as a 32–bit value in an AIX operating system. If a DOS program provides an integer constant for this argument, the Microsoft C compiler puts this constant in a 16–bit **int** value by default. The compiler then generates code that passes this 16–bit value to a procedure that is expecting a 32–bit (**key_t**) value.

The following examples illustrate correct and incorrect use of argument types:

```
return = msgget((key_t)KEY,msgflg);      (correct: KEY is 32 bit value)
return = msgget(KEY,msgflg);             (incorrect: KEY is 16 bit value)
```

Refer to the manual page for the appropriate function to verify that the arguments used match those expected by the function being called.

Using PCILIB Polling Calls

Applications developed using PCILIB will run under both the Access program and DOS Merge if appropriate programming techniques are used. This section describes how to ensure that applications using message queues, semaphores, or **uwait** calls are developed properly.

Under an AIX operating system, when a program makes a call to read a message queue, the program may specify that the call should wait for a message to be placed on a message queue before returning. This is referred to as a blocking call. Under DOS, however, blocking message operations are not supported. Thus, applications using PCILIB have to make repeated calls to check for messages until a message becomes available. This checking is referred to as *polling*.

Constant polling may produce a heavy load on the host, resulting in degraded system performance. In order to reduce the load and improve system performance, the application should alternate polling with sleeping. PCILIB provides a **sleep_h** function to facilitate this. During the inactivity produced by the sleep time, other processes are able to use the processors in the host or personal computer environment.

In choosing the period of time to use for sleep intervals, consider the following general statements:

1. The longer the interval, the lower the load on the host.

2. The shorter the interval, the faster the response time.

Note, however, that frequent polling increases the load on the host, possibly slowing response time. An appropriate compromise must be reached.

These comments apply to semaphore polling and **uwait** polling as well as message queue polling.

Special Considerations for DOS Merge

In developing a PCILIB application to run under DOS Merge, in addition to the factors listed above, sleep granularity must be considered. Sleep granularity—that is, the time interval between ticks of the system clock—may be different in the DOS Merge environment than in a personal computer. DOS clock interrupts normally occur 18.2 times per second, whereas DOS Merge clock interrupts normally occur once per second.

PCILIB sleep calls, when given a value corresponding to *n* clock–ticks, generally sleep for between *n* –1 and *n* clock–tick intervals. For example, a value corresponding to two clock–ticks results in sleeping between one and two clock–tick intervals. Thus, under DOS Merge (where the default clock–tick interval is one second) the minimum sleep interval ranges between zero to one second. Under DOS, the minimum sleep interval ranges between zero to 1/18.2 seconds.

If the **fastclk** option is enabled under DOS Merge, the clock rate becomes that of a normal personal computer, approximately 18.2 times per second. This provides finer control of the sleep interval, though the latency is still subject to AIX operating system constraints. Because delivering the extra clock interrupts takes CPU time, running **fastclk** may improve the response time when the system is not loaded but worsen it when other tasks are competing for CPU usage.

fastclk may improve the latency of a DOS task but leads to a higher overall load on the system. An application using the **fastclk** option should be tested under loaded conditions to see if it improves performance.

Refer to the **sleep_h** manual page for details on the use of **sleep_h** and **fastclk**.

SEE ALSO

dflthost
msgctl
msgget
msgrcv
msgsnd
semctl
semget
semop
sleep_h

NAME

lcs_intro – introduction to language character set routines

SYNOPSIS

This section describes all of the language character set library routines.

DESCRIPTION

The **lcs** routines translate from one code set or code page to another so that a viewable character (glyph) under one code page or code set is viewable under another code page or code set.

Certain glyphs cannot be translated exactly. However, they can be represented by multiple glyphs. Other glyphs may have no possible representation, in which case a user default character or a best single character is used (if nothing better is available, the best single character is an asterisk). The order of precedence during glyph translation is given below:

- Exact — The glyph is translated to the exact same glyph in the different character set.
- Multiglyph — The glyph is translated to a multiglyph output that is a best translation possible.

  For example, in some code sets there is a three-quarters glyph (¾). It may not be possible to translate that glyph exactly, but it is possible to display the '3' glyph, the '/' glyph, and the '4' glyph, in sequence, to convey the same meaning.
- User char — There was no exact translation possible, so a user-supplied character was used instead.
- Best single — The best single glyph translation was used. There was no user character defined. If nothing better is available, the default is an asterisk (*).

Most of these routines return one or more error conditions. An error condition is indicated by an otherwise impossible returned value, such as −1 or the NULL pointer; the individual manual pages specify the details. An error number is also made available in the external variable **lcs_errno**. **lcs_errno** is not always cleared on successful calls, so you should test it only after an error has been indicated.

The following errors are returned:

LCS_ERR_NOTFOUND	The translation table was not found.
LCS_ERR_BADTABLE	The translation table is invalid.

LCS_ERR_NOTABLE	Translation tables are not set.
LCS_ERR_NOSPACE	There is insufficient space in the output buffer.
LCS_ERR_STOPXLAT	An untranslatable character was seen, and the stop-translation option mode bit was set.
LCS_ERR_INPUT_SPLIT	The translation routines encountered a byte that could be part of a multibyte character in the input buffer. The portion of the block prior to this character was correctly translated. The remaining byte should be added to the next block for translation. This normally occurs in packet network situations where a block of text could be sent as several discrete packets requiring some reassembly.

Additionally, the following six global variables provide translation statistics:

int lcs_exact_translations;

This variable contains the number of characters that were translated exactly. Exact translation means that the same glyph exists in both the input translation table and the output translation table.

int lcs_multiple_translations;

This variable contains the number of characters that generated multiple output glyphs for a single input glyph. This form of translation does not happen if the LCS_MODE_NO_MULTIPLE option bit is set.

int lcs_best_single_translations;

This variable contains the number of characters for which there was no exact translation and no multiple glyph translation. The option bits LCS_MODE_STOP_XLAT and LCS_MODE_USER_CHAR must both be reset for this form of translation to occur.

int lcs_user_default_translations;

This variable contains the number of user-specified default character translations. This form of translation happens when there is no exact translation, no multiple translation, the LCS_MODE_STOP_XLAT option bit is reset, and the LCS_MODE_USER_CHAR option bit is set.

int lcs_input_bytes_processed;

This variable contains the number of bytes processed from the input buffer. It is valid regardless of whether the translate function succeeded or not.

int lcs_output_bytes_processed;

This variable contains the number of bytes placed into the output buffer. It is valid regardless of whether the translate function succeeded or not.

These six variables are reset with each **lcs_translate_string()** and **lcs_translate_block()** library call.

NAME

dflthost – get/set the current host for PCILIB IPC functions

SYNOPSIS

```
#include <pcilib.h>
#include <memmdl.h>

int dflthost( drvnum )
int drvnum
```

DESCRIPTION

dflthost allows the user to read and/or set which host is used for performing PCILIB IPC operations. **dflthost** selects the host corresponding to the virtual drive indicated by **drvnum** (A: = 1, B: = 2, etc.) (This virtual drive may or may not be logged into a host.) Initially, the host corresponding to the first virtual drive is selected.

If **drvnum** is 0, or any value that does not correspond to a virtual drive, then the current default host is not changed. This option can be used to get the current default host.

If a user program changes the default host, it should restore it to its original value before exiting.

RETURN VALUES

dflthost always returns what the current default host was before the call was made. The return value is an integer assigned as A: = 1, B: = 2, etc.

EXAMPLE

```
        :
        :
int lastHost, setHost;
        :
        :
/*
** Change Current Default Host
** to drive D: (4)
*/
setHost = 4;
if ( (lastHost = dflthost(setHost)) == 0 )
   errorHandler();
else
   printf("Previous Host = %d, Current Host = %d", lastHost, setHost);
```

SEE ALSO

msgctl
msgget
msgrcv
msgsnd
semctl
semget
semop

NAME

feature, feature_exists, feature_list – get features supported by a virtual drive

SYNOPSIS

```
#include <pcilib.h>
#include <memmdl.h>

char *feature_list(drive)
int drive;

char *feature(drive, token)
int drive;
char *token;

int *feature_exists(drive, token, version)
int drive;
char *token, *version;
```

DESCRIPTION

feature_list() returns a pointer to a null-terminated string containing feature version information of the form:

```
"token=version[,token=version...]"
```

For example, the feature list could appear as:

```
"F=1,PCI=3.2.0,LCSTBL=8859"
```

feature() calls **feature_list()** as a subroutine, and returns a pointer to a null-terminated version string corresponding to a "token=version" pair in the string returned by **feature_list()**.

feature_exists() calls **feature()** as a subroutine and returns 1 if the specified token is paired with the specified version in the string returned by **feature()**; otherwise, it returns 0. If called with version set to NULL, **feature_exists()** returns 1 if the specified token exists at all in the string returned by **feature_list()**.

drive is the drive number of the host whose feature list is being requested. Drive letters and numbers correspond in the following way: A: = 1, B: = 2, and so forth. A value of 0 indicates the local bridge.

In the Access program Version 2.1 or greater, the **LCSTBL** feature is present and returns the AIX operating system code set associated with the virtual drive. The feature **PCI** returns the Access program version number of the local bridge. This can also be used to determine if an application is running under the Access program or DOS Merge.

Any other features not documented here are for internal use only.

DIAGNOSTICS

If the server specified does not support feature-version list swapping, **feature_list()** returns a pointer to a null string (""). If the drive specified does not refer to the local bridge device or a connected remote server, or if the local bridge does not support feature–version list swapping, **feature_list()** returns NULL.

If the token specified cannot be found in the string returned by **feature_list()**, or if **feature_list()** returns NULL for the drive specified, **feature()** returns NULL.

If **feature()** returns NULL for the drive and token specified, **feature_exists()** returns 0.

CAVEATS

The return values of **feature()** and **feature_list()** point to internal data areas that should not be corrupted. Furthermore, these pointers should not be relied upon after subsequent calls to any of these routines.

EXAMPLE

Using the sample program below, the following queries on a virtual drive connected to a system running the Access program 2.1 would display the responses shown:

```
Query:      feature d: PCI
Response:   Feature PCI is 3.2.0

Query:      feature d:
Response:   Feature list of D: is F=1,PCI=3.2.0,LCSTBL=8859

Query:      feature 0 MERGE
Response:   Feature MERGE does not exist

Query:      feature 0 PCI 3.2.0
Response:   Feature PCI=3.2.0 exists
```

Source code:

```
#include <stdio.h>
#include <ctype.h>
#include <pcilib.h>

main(argc, argv)
int argc;
char **argv;
{
    char *res;
    int ires;
    int host;
    char *token;
    char *vers;

    if (argc < 2) {
        printf("usage: feature host [token [vers]]\n");
        exit(1);
    }
    host = 0;
    token = NULL;
    vers = NULL;
    argv++;
    if (isdigit(**argv))
        sscanf(*argv, "%d", &host);
    else
        host = tolower(**argv) - 'a' + 1;
    if (argc >= 3)
        token = *++argv;
    if (argc >= 4) {
        vers = *++argv;
        ires = feature_exists(host, token, vers);
        printf("Feature %s=%s %s\n", token, vers,
            ires ? "exists" : "does not exist");
        exit(0);
    }
    if (argc == 3) {
        res = feature(host, token);
        if (res != NULL)
            printf("Feature %s is %s\n", token, res);
        else
            printf("Feature %s does not exist\n", token);
    } else {
        res = feature_list(host);
        if (res != NULL)
            printf("Feature list of %c: is %s\n", host+'@', res);
        else
            printf("Feature list does not exist on %c\n", host+'@');
    }
}
```

NAME

getuattr – get AIX operating system file attributes

SYNOPSIS

```
#include <pcilib.h>
#include <memmdl.h>

int getuattr( upath )
char *upath;
```

DESCRIPTION

getuattr is a DOS function used to determine the AIX operating system file attributes of the specified AIX operating system path.

upath is the AIX operating system path name whose attributes are being requested. The path may include a drive designation (for example, D:) indicating on which host the file is stored. If no drive is specified, the current drive is used.

RETURN VALUES

If the call succeeds, the AIX operating system file mode is returned (see **stat** in the *AIX Operating System Technical Reference Manual*).

If the call fails for any reason, a value of –1 is returned.

EXAMPLE

```
        :
        :
char *uPath = "D:/existing/\*(un/file";
int  uAttrib;
        :
        :
/*
** Get attributes of the AIX file
** on virtual drive D:
*/
if ( (uAttrib = getuattr(uPath)) == -1 )
        errorHandler();
else
        printf("Attribute - in octal - is: %o", uAttrib);

        :
        :
```

SEE ALSO

isvirtual
uchmod

NAME

isvirtual – return the virtual drive number of a specified path

SYNOPSIS

```
#include <pcilib.h>
#include <memmdl.h>

int isvirtual( dpath )
char *dpath;
```

DESCRIPTION

isvirtual is a DOS function that returns the virtual drive number of the specified path.

dpath is the DOS path name whose virtual drive location is to be determined. The pathname **dpath** need not refer to an existing file.

RETURN VALUES

If the specified DOS path is found on a virtual drive, the drive number of the device is returned. Drive A returns as 1, drive B returns as 2, and so forth.

If the specified path is not on a virtual drive, a value of 0 returns.

EXAMPLE

```
        :
        :
char *dPath = "D:\\DOS\\FILENAME";
int  driveNum;
        :
        :

/*
** Check if the DOS path name is
** on a virtual drive
*/
if ( (driveNum = isvirtual(dPath)) == 0 )
     printf("%s is NOT on a virtual drive.\n", dPath);
else
     printf("The DOS Drive Number for %s is: %d", dPath, driveNum);

        :
        :
```

SEE ALSO

vdrive

NAME

lcs_get_table – get language character set conversion table

SYNOPSIS

```
#include <lcs.h>

lcs_tbl lcs_get_table(table, lcspath_default)
          char *table, *lcspath_default;
```

DESCRIPTION

This function opens a translation table that the other language character set functions can use. It returns NULL if the table is not available. The LCSPATH environment variable specifies the directory where the translation tables are stored. If the environment variable is not found, **lcspath_default** is used. The format of the file opened is $LCSPATH/*table*.lcs. LCSPATH can contain multiple directories separated by semicolons.

RETURN VALUES

On successful completion, an **lcs_tbl** is returned. NULL is returned if the table is not available. Error codes are returned in **lcs_errno**.

EXAMPLE

```
#include <lcs.h>
#define LCSPATH_DEFAULT "c:\\pci\\lib;c:\\usr\\lib\\merge\\lcs"
lcs_tbl input_table;
char *input_tbl_name="pc437";/* use DOS code page 437 */
      :
      :
/* Set input table */
if ((input_table = lcs_get_table(input_tbl_name,
                   LCSPATH_DEFAULT)) == NULL) {
      errorHandler();
}
      :
      :
```

SEE ALSO

lcs_intro

NAME

lcs_release_table – releases a language character set conversion table

SYNOPSIS

```
#include <lcs.h>

int lcs_release_table(table)
          lcs_tbl table;
```

DESCRIPTION

This function frees the memory that was previously allocated when **lcs_get_table** was called. The **lcs_tbl** that is being released is no longer valid.

RETURN VALUES

This function returns 0 on success and –1 when the table is invalid. Error codes are returned in **lcs_errno**.

EXAMPLE

```
#include <lcs.h>
lcs_tbl input_table;
      :
      :
/* Free up the lcs table */
if (lcs_release_table(input_table) < 0 )  {
      errorHandler();
}
```

SEE ALSO

lcs_intro

NAME

lcs_set_options – sets language character set conversion table options

SYNOPSIS

```
#include <lcs.h>

void lcs_set_options(mode, default_char, country)
          unsigned short mode;
          unsigned short default_char;
          short country;
```

DESCRIPTION

The **mode** argument is a bitmap of translation options:

LCS_MODE_NO_MULTIPLE	Don't translate a character into multiple glyphs.
LCS_MODE_STOP_XLAT	Stop translating when a character that cannot be translated exactly is encountered.
LCS_MODE_USER_CHAR	Use the user-specified **default_char** for translation failures.
LCS_MODE_UPPERCASE	Perform the lower-to-uppercase translation as well as character set translation.
LCS_MODE_LOWERCASE	Perform the upper-to-lowercase translation as well as character set translation.

When an untranslatable character is encountered and the LCS_MODE_STOP_XLAT bit is not set, a best single-byte translation is used.

When both of the options LCS_MODE_UPPERCASE and LCS_MODE_LOWERCASE are specified, the upper-to-lowercase translation is performed.

The default **mode** is 0.

The **default_char** argument is the character to use when an untranslatable character is encountered if the options LCS_MODE_USER_CHAR and LCS_MODE_NO_MULTIPLE are set. If the LCS_MODE_NO_MULTIPLE option is not set and the LCS_MODE_USER_CHAR option is set, an untranslatable character results in a multibyte character if one exists or in **default_char** if no multibyte character exists.

The **country** option is used for country-specific modifications to the upper– and lowercase translations. This should be a DOS country number. If the country number is not recognized by the system, no modifications are made.

The default **country** is 1 (US).

EXAMPLE

```
#include <lcs.h>
/* use our own default character,
 * rather than '*'.
 * Also, don't allow multiple character
 * translations (e.g.: don't translate
 * the ISO 8859-1 three-quarters symbol
 * into the three byte '3/4').
 */
short mode = LCS_MODE_USER_CHAR | LCS_MODE_NO_MULTIPLE;
unsigned short default_char = '!';/* use ! for default character */
short country = 1;          /* country code 1 is US */
      :
      :
/* Now that we've got our tables,
 * set the options.
 */
lcs_set_options(mode, default_char, country);
      :
      :
```

SEE ALSO

lcs_intro

NAME

lcs_set_tables – sets input and output language character set conversion tables

SYNOPSIS

```
#include <lcs.h>

int lcs_set_tables(out_tbl, in_tbl)
          lcs_tbl out_tbl, in_tbl;
```

DESCRIPTION

This function sets the translation tables for the conversion functions. The tables must first be obtained by calling **lcs_get_table**.

RETURN VALUES

This function returns 0 on success and −1 if one of the tables is invalid. Error codes are returned in **lcs_errno**.

EXAMPLE

```
#include <lcs.h>
lcs_tbl input_table;
lcs_tbl output_table;
      :
      :
/* Get input table */
      :
      :
/* Get output table */
      :
      :
/* Set the tables */
if ( lcs_set_tables(output_table, input_table) < 0 ) {
      errorHandler();
}
      :
      :
```

SEE ALSO

lcs_intro
lcs_get_table

NAME

lcs_translate_block – translate input block to output block

SYNOPSIS

```
#include <lcs.h>

int lcs_translate_block(out, out_size, in, in_len)
        char *out, *in;
        int out_size, in_len;
```

DESCRIPTION

This function translates a block of characters in **in** and places the results in **out**. **in_len** specifies the size in bytes of **in**. No more than **out_size** bytes are placed in **out**. **lcs_translate_block**, unlike **lcs_translate_string**, does not recognize '\0' as a delimiter, which means multistring text blocks get translated.

To use this function, you must first get the tables, set the tables, and set the options.

RETURN VALUES

If the translation is successful, this function returns the number of bytes translated into **out**. It returns −1 when a translation error occurs. Translation statistics are described in **lcs_intro**. Error codes are returned in **lcs_errno**.

EXAMPLE

```
#include <lcs.h>
#define MAX_LINE 1024;
char inbuf[MAX_LINE];
char outbuf[2*MAX_LINE];  /* allow us to double our output size */
      :
      :
/* Get input table */
      :
      :
/* Get output table */
      :
      :
/* Set the tables */
      :
      :
/* Set the options */
      :
      :
if ( lcs_translate_block(outbuf, 2*MAX_LINE, inbuf, MAX_LINE) < 0) {
      switch(lcs_errno) {
```

```
        case LCS_ERR_NOTFOUND:
                fprintf(stderr,"Translation table(s) not found!\n");
                exit(1);
        case LCS_ERR_BADTABLE:
                fprintf(stderr, "Translation table(s) bad!\n");
                exit(1);
        case LCS_ERR_NOTABLE:
                fprintf(stderr, "Translation tables not set!\n");
                exit(1);
        case LCS_ERR_NOSPACE:
                /* Insufficient space, hhmm */
                fprintf(stderr,
                        "Insufficient space in output buffer!\n");
                exit(1);
        case LCS_ERR_STOPXLAT:
                /* Untranslatable character and abort_on */
                fprintf(stderr, "Untranslatable character!\n");
                exit(1);
        case LCS_ERR_INPUT_SPLIT:
                /* Input is split across multiple buffers */
                fprintf(stderr, "Input split!\n");
                exit(1);
        default:
                /* Unknown error condition, help! */
                fprintf(stderr, "Unknown error: %d\n", lcs_errno);
                exit(1);
        } /* end switch */
}
```

SEE ALSO

lcs_intro
lcs_translate_string
lcs_get_table
lcs_set_tables
lcs_set_options

NAME

lcs_translate_string – translates input string to output string

SYNOPSIS

```
#include <lcs.h>

int lcs_translate_string(out, out_size, in)
            char *out, *in;
            int out_size;
```

DESCRIPTION

This function translates a NULL-terminated string of characters in **in** and places the result in **out**. No more than **out_size** characters, including a terminating NULL, are placed in **out**.

To use this function, you must first get the tables, set the tables, and set the options.

RETURN VALUES

This function returns 0 on a successful translation and –1 when a translation error occurs. Translation statistics are described in **lcs_intro**. Error codes are returned in **lcs_errno**.

EXAMPLE

```
#include <lcs.h>
#define MAX_LINE 1024;
char inbuf[MAX_LINE];
char outbuf[MAX_LINE];
       :
       :
/* Get input table */
       :
       :
/* Get output table */
       :
       :
/* Set the tables */
       :
       :
/* Set the options. */
       :
       :
if ( lcs_translate_string(outbuf, MAX_LINE, inbuf) < 0) {
       switch(lcs_errno) {
       case LCS_ERR_NOTFOUND:
              fprintf(stderr,"Translation table(s) not found!\n");
              exit(1);
```

```
        case LCS_ERR_BADTABLE:
               fprintf(stderr, "Translation table(s) bad!\n");
               exit(1);
        case LCS_ERR_NOTABLE:
               fprintf(stderr, "Translation tables not set!\n");
               exit(1);
        case LCS_ERR_NOSPACE:
               /* Insufficient space, hhmm */
               fprintf(stderr,
                     "Insufficient space in output buffer!\n");
               exit(1);
        case LCS_ERR_STOPXLAT:
               /* Untranslatable character and abort_on */
               fprintf(stderr, "Untranslatable character!\n");
               exit(1);
        case LCS_ERR_INPUT_SPLIT:
               /* Input is split across multiple buffers */
               fprintf(stderr, "Input split!\n");
               exit(1);
        default:
               /* Unknown error condition, help! */
               fprintf(stderr, "Unknown error: %d\n", lcs_errno);
               exit(1);
        } /* end switch */
}
```

SEE ALSO

lcs_intro
lcs_translate_block
lcs_get_table
lcs_set_tables
lcs_set_options

NAME

mapd2u – map a DOS path name to an AIX operating system path name

SYNOPSIS

```
#include <pcilib.h>
#include <memmdl.h>

int mapd2u( upath, dpath )
char *upath, *dpath;
```

DESCRIPTION

mapd2u is a DOS function that returns, in AIX operating system path name conventions, the fully qualified translation of a DOS path name.

upath points to the AIX operating system path name supplied by the function. It is a fully qualified, null-terminated string with a drive designation that exactly corresponds to the DOS path.

dpath is the DOS path name. It must be provided as a null-terminated string.

RETURN VALUES

On successful completion, a value of 0 is returned.

If the translation fails (the DOS file name does not exist), a value of −1 returns and the DOS error code is returned in **errno**. The translation may fail if the DOS file name contains a single quote (').

EXAMPLE

```
        :
        :
char uPath[128];
char dPath[128] = "D:\\DOS\\PATHNAME";
int  retVal;
        :
        :
/* Find the AIX name for the DOS path on virtual drive D:
** Note: If the DOS name has a drive designation, the returned AIX
** path name will also have one.  (In this example it will be: D:)
*/
if ( (retVal = mapd2u(uPath, dPath)) == -1 )
     errorHandler();
else
     printf("Mapped AIX name - including drive designation: %s",uPath);
        :
        :
```

SEE ALSO

isvirtual
mapu2d

BUGS

No error is returned if a path component does not exist. The nonexistent component and remainder of the path are simply forced to lowercase. The calling program must ensure that the path exists before calling this function.

NAME

mapu2d – map an AIX operating system pathname to a DOS pathname

SYNOPSIS

```
#include <pcilib.h>
#include <memmdl.h>

int mapu2d( dpath, upath )
char *dpath, *upath;
```

DESCRIPTION

mapu2d is a DOS function that returns, in DOS path name conventions, the translation of an AIX operating system path name.

dpath is the DOS name returned by the function. It is a fully qualified (including driveletter), null-terminated string that exactly corresponds to the AIX operating system path.

upath is the AIX operating system path name. It must be provided as a null-terminated string. The path name may include a drive designation to indicate which host the translation is performed on. If no drive is specified, the current drive is used. The **upath** parameter uses normal AIX operating system conventions (for example, forward slashes as separators).

RETURN VALUES

On successful completion, a value of 0 returns.

If the translation fails, a value of −1 returns. Possible causes for failure are: the AIX operating system file name does not exist; access is denied; or the path contains an invalid component.

EXAMPLE

```
        :
        :
char uPath[128] = "D:/aix/pathname";
char dPath[128];
int  retVal;
        :
        :
/* Find the DOS name for the aix path on virtual drive D: */
if ( (retVal = mapu2d(dPath, uPath)) == -1 )
        errorHandler();
else
        printf("Mapped DOS name is: %s", dPath);
        :
        :
```

SEE ALSO

isvirtual
mapd2u

BUGS

No error is returned if a path component does not exist. The nonexistent component and remainder of the path is simply forced to uppercase. The calling program must ensure that the path exists before calling this function.

NAME

msgctl – message control operations

SYNOPSIS

```
#include <pcilib.h>
#include <memmdl.h>
#include <ipc.h>

int msgctl (msqid, cmd, buf)
int msqid, cmd;
struct msqid_ds *buf;
```

DESCRIPTION

msgctl provides a variety of message control operations as specified by **cmd**. The operations are performed on the current host (as specified in the most recent **dflthost** call). The following **cmd**s are available:

IPC_STAT Place the current value of each member of the data structure associated with **msqid** into the structure pointed to by **buf**. The contents of this structure are defined in **intro**.

IPC_SET Set the value of the following members of the data structure associated with **msqid** to the corresponding value found in the structure pointed to by **buf**:

```
msg_perm.uid
msg_perm.gid
msg_perm.mode        /* only low nine bits */
msg_qbytes
```

This **cmd** can only be executed by a process with an effective user ID equal to either the super-user or the value of **msg_perm.[c]uid** in the data structure associated with **msqid**. Only the super-user can raise the value of **msg_qbytes**.

IPC_RMID Remove the message queue identifier specified by **msqid** from the AIX operating system and destroy the message queue and data structure associated with it. This **cmd** can only be executed by a process that has an effective user ID equal to either that of super-user or to the value of **msg_perm.[c]uid** in the data structure associated with **msqid**.

msgctl fails if one or more of the following are true:

[EINVAL] **msqid** is not a valid message queue identifier.

[EINVAL] **cmd** is not a valid command.

[EACCES] **cmd** is equal to **IPC_STAT** and read operation permission is denied to the calling process (see **intro**).

[EPERM] **cmd** is equal to **IPC_RMID** or **IPC_SET**, and the effective user ID of the calling process is not equal to that of super-user and is not equal to the value of **msg_perm.[c]uid** in the data structure associated with **msqid**.

[EPERM] **cmd** is equal to **IPC_SET**, an attempt is being made to increase to the value of **msg_qbytes**, and the effective user ID of the calling process is not equal to that of super-user.

[EFAULT] **buf** points to an illegal address.

RETURN VALUE

On successful completion, a value of 0 returns. If the call fails for any reason, a value of −1 returns and **errno** is set to indicate the error.

SEE ALSO

dflthost
intro
msgget
msgrcv
msgsnd

NAME

msgget – get message queue

SYNOPSIS

```
#include <pcilib.h>
#include <memmdl.h>
#include <ipc.h>

int msgget (key, msgflg)
key_t key;
int msgflg;
```

DESCRIPTION

msgget returns the message queue identifier associated with **key** from the AIX operating system host specified in the most recent **dflthost** call.

A message queue identifier and associated message queue and data structure (see **intro**) are created for **key** if one of the following is true:

key is equal to **IPC_PRIVATE.**

key does not already have a message queue identifier associated with it, and **(msgflg & IPC_CREAT)** is "true."

Upon creation, the data structure associated with the new message queue identifier is initialized as follows:

Msg_perm.cuid, msg_perm.uid, msg_perm.cgid, and **msg_perm.gid** are set to the effective user ID and effective group IDs, respectively, of the calling process.

The low-order nine bits of **msg_perm.mode** are set equal to the low-order nine bits of **msgflg.**

msg_qnum, msg_lspid, msg_lrpid, msg_stime, and **msg_rtime** are set to 0.

msg_ctime is set to the current AIX operating system time.

msg_qbytes is set to the AIX operating system limit.

msgget fails if one or more of the following are true:

[EACCES] A message queue identifier exists for **key**, but operation permission (see **intro**) as specified by the low-order nine bits of **msgflg** would not be granted.

[ENOENT] A message queue identifier does not exist for **key**, and (**msgflg & IPC_CREAT**) is "false."

[ENOSPC] A message queue identifier is to be created, but the number of message queue identifiers would exceed the system maximum allowed for that host.

[EEXIST] A message queue identifier exists for **key**, but (**msgflg & IPC_CREAT**) **& (msgflg & IPC_EXCL)** is "true."

RETURN VALUES

On successful completion, a message queue identifier is returned. Otherwise, a value of −1 returns and **errno** is set to indicate the error.

SEE ALSO

dflthost
intro
msgctl
msgrcv
msgsnd

NAME

msgrcv – message receiving

SYNOPSIS

```
#include <pcilib.h>
#include <memmdl.h>
#include <ipc.h>

int msgrcv (msqid, msgp, msgsz, msgtyp, msgflg)
int msqid;
struct msgbuf *msgp;
int msgsz;
long msgtyp;
int msgflg;
```

DESCRIPTION

msgrcv reads a message from the queue associated with the message queue identifier specified by **msqid** and places it in the structure pointed to by **msgp**. This structure is composed of the following members:

```
long mtype;      /* message type */
char mtext[];    /* message text */
```

mtype is the type of the received message as specified by the sending process. **mtext** is the text of the message. **msgsz** specifies the size of **mtext**, which can be no greater than 1008. The received message is truncated to **msgsz** bytes if it is larger than **msgsz** and **(msgflg & MSG_NOERROR)** is "true." The truncated part of the message is lost, and no indication of the truncation is given to the calling process.

msgtyp specifies the type of message requested, as follows:

- If **msgtyp** is equal to 0, the first message on the queue is received.
- If **msgtyp** is greater than 0, the first message of type **msgtyp** is received.
- If **msgtyp** is less than 0, the first message of the lowest type that is less than or equal to the absolute value of **msgtyp** is received.

msgflg specifies the action to be taken if a message of the desired type is not on the queue, as follows:

- If **(msgflg & IPC_NOWAIT)** is "true," the calling process immediately returns a value of –1 and **errno** is set to ENOMSG.

- If **(msgflg & IPC_NOWAIT)** is "false," the calling function returns a value of −1 and **errno** is set to EIDRM.

msgrcv fails and no message is received if one or more of the following are true:

[EINVAL] **msqid** is not a valid message queue identifier.

[EACCES] Operation permission is denied to the calling process.

[EINVAL] **msgsz** is less than 0.

[E2BIG] **mtext** is greater than **msgsz** and **(msgflg & MSG_NOERROR)** is "false."

[ENOMSG] The queue does not contain a message of the desired type and **(msgtyp & IPC_NOWAIT)** is "true."

[EFAULT] **msgp** points to an illegal address.

On successful completion, the following actions are taken with respect to the data structure associated with **msqid** (see **intro**).

- **msg_qnum** is decremented by 1.
- **msg_lrpid** is set to the process ID of the calling process.
- **msg_rtime** is set to the current time.

RETURN VALUES

On successful completion, **msgrcv** returns the number of bytes actually placed into **mtext**.

If **msgrcv** returns due to removal of **msqid** from the system, a value of −1 returns and **errno** is set to EIDRM.

Otherwise, a value of −1 returns and **errno** is set to indicate the error.

SEE ALSO

dflthost
intro
msgctl
msgget
msgsnd
sleep_h

DIFFERENCE FROM SYSTEM V

msgrcv calls that would cause a process to suspend in System V-compatible operating system cause an error to be returned in DOS. DOS tasks may not be suspended using **msgrcv** functions; **sleep_h** may be used instead.

NAME

msgsnd – message sending

SYNOPSIS

```
#include <pcilib.h>
#include <memmdl.h>
#include <ipc.h>

int msgsnd (msqid, msgp, msgsz, msgflg)
int msqid;
struct msgbuf *msgp;
int msgsz, msgflg;
```

DESCRIPTION

msgsnd sends a message to the queue associated with the message queue identifier specified by **msqid**. The operations are performed on the current host (as specified in the most recent **dflthost** call). **msgp** points to a structure containing the message. This structure is composed of the following members:

```
long mtype;      /* message type */
char mtext[];    /* message text */
```

mtype is a positive integer that can be used by the receiving process for message selection. **mtext** is any text of length **msgsz** bytes. **msgsz** ranges from zero to a system-imposed maximum which can be no greater than 1008.

msgflg specifies the action to be taken if either or both of the following are true:

- The number of bytes already on the queue is equal to **msg_qbytes**.
- The total number of messages on all queues system-wide is equal to the AIX operating system-imposed limit.

If one of the two events occurs and **msgflg** is "true," the message is not sent and the DOS function returns immediately. If **msgflg** is "false," the DOS function returns −1 and **errno** is set to EIDRM. Under no circumstances does the PCILIB **msgsnd** function allow the DOS task to suspend.

msgsnd fails and no message is sent if one or more of the following are true:

[EINVAL] **msqid** is not a valid message queue identifier.

[EACCES] Operation permission is denied to the calling process (see **intro**).

[EINVAL] **mtype** is less than 1.

[EAGAIN] The message cannot be sent for one of the reasons cited above, and **(msgflg & C_NOWAIT)** is "true."

[EINVAL] **msgsz** is less than zero or greater than the system-imposed limit.

[E2BIG] **msgsnd** is greater than 1008. **mtext** is greater than **msgsz** and **(msgflg & MSG_NOERROR)** is "false."

[EFAULT] **msgp** points to an illegal address.

On successful completion, the following actions are taken with respect to the data structure associated with **msqid** (see **intro**).

- **msg_qnum** is incremented by 1.
- **msg_lspid** is set to the process ID of the calling process.
- **msg_stime** is set to the current time.

RETURN VALUES

On successful completion, **msgsnd** returns a value of 0.

If **msgsnd** returns due to removal of **msqid** from the system, a value of −1 returns and **errno** is set to EIDRM.

Otherwise, a value of −1 returns and **errno** is set to indicate the error.

SEE ALSO

dflthost
intro
msgctl
msgget
msgrcv
sleep_h

DIFFERENCE FROM SYSTEM V

msgsnd calls that would cause a process to suspend in System V-compatible operating system cause an error to be returned in DOS. DOS tasks may not be suspended using **msgsnd** functions; **sleep_h** may be used instead.

NAME

semctl – semaphore control operations

SYNOPSIS

```
#include <pcilib.h>
#include <memmdl.h>
#include <ipc.h>

int semctl (semid, semnum, cmd, arg)
int semid, semnum, cmd;
union semun {
     int val;
     struct semid_ds *buf;
     ushort *array;
} arg;
```

DESCRIPTION

semctl provides a variety of semaphore control operations as specified by **cmd**. The operations are performed on the current host (as specified in the most recent **dflthost** call).

The following **cmd**s are executed with respect to the semaphore specified by **semid** and **semnum**:

GETVAL Return the value of **semval** (see **intro**).

SETVAL Set the value of **semval** to **arg.val**. When this **cmd** is successfully executed, the **semadj** value corresponding to the specified semaphore in all AIX operating system processes is cleared.

GETPID Return the value of **sempid**.

GETNCNT Return the value of **semncnt**.

GETZCNT Return the value of **semzcnt**.

The following **cmd**s return and set, respectively, every **semval** in the set of semaphores.

GETALL Place **semval**s into the array pointed to by **arg.array**.

SETALL Set **semval**s according to the array pointed to by **arg.array**. When this **cmd** is successfully executed, the **semadj** values corresponding to each specified semaphore in all AIX operating system processes are cleared.

The following **cmd**s are also available:

IPC_STAT Place the current value of each member of the data structure associated with **semid** into the structure pointed to by **arg.buf**. The contents of this structure are defined in **intro**.

IPC_SET Set the value of the following members of the data structure associated with **semid** to the corresponding value found in the structure pointed to by **arg.buf**:

```
sem_perm.uid
sem_perm.gid
sem_perm.mode        /* only low nine bits */
```

This **cmd** can only be executed by a process that has an effective user ID equal to either that of super-user or to the value of **sem_perm.[c]uid** in the data structure associated with **semid**.

IPC_RMID Remove the semaphore identifier specified by **semid** from an AIX operating system and destroy the set of semaphores and data structure associated with it. This **cmd** can only be executed by a process that has an effective user ID equal to either that of super-user or to the value of **sem_perm.[c]uid** in the data structure associated with **semid**.

semctl fails if one or more of the following are true:

[EINVAL] **semid** is not a valid semaphore identifier.

[EINVAL] **semnum** is less than zero or greater than **sem_nsems**.

[EINVAL] **cmd** is not a valid command.

[EACCES] Operation permission is denied to the calling process (see **intro**).

[ERANGE] **cmd** is **SETVAL** or **SETALL** and the value to which **semval** is to be set is greater than the AIX operating system-imposed maximum.

[EPERM] **cmd** is equal to **IPC_RMID** or **IPC_SET**, and the effective user ID of the calling process is not equal to that of super-user, and it is not equal to the value of **sem_perm.[c]uid** in the data structure associated with **semid**.

[EFAULT] **arg.buf** points to an illegal address.

RETURN VALUES

On successful completion, the value returned depends on **cmd** as follows:

GETVAL is the value of **semval**.

GETPID is the value of **sempid**.

GETNCNT is the value of **semncnt**.

GETZCNT is the value of **semzcnt**.

All others have a value of 0.

If the operation fails, a value of −1 returns and **errno** is set to indicate the error.

SEE ALSO

dflthost
intro
semget
semop

NAME

semget – get set of semaphores

SYNOPSIS

```
#include <pcilib.h>
#include <memmdl.h>
#include <ipc.h>

int semget (key, nsems, semflg)
key_t key;
int nsems, semflg;
```

DESCRIPTION

semget returns the semaphore identifier associated with **key**. The operations are performed on the current host (as specified in the most recent **dflthost** call).

A semaphore identifier and associated data structure and set containing **nsems** semaphores (see **intro**) are created for **key** if one of the following is true:

- **key** is equal to **IPC_PRIVATE.**
- **key** does not already have a semaphore identifier associated with it, and **(semflg & IPC_CREAT)** is "true."

On creation, the data structure associated with the new semaphore identifier is initialized as follows:

- **sem_perm.cuid**, **sem_perm.uid**, **sem_perm.cgid**, and **sem_perm.gid** are set to the effective user ID and effective group IDs, respectively, of the calling process.
- The low-order nine bits of **sem_perm.mode** are set equal to the low-order nine bits of **semflg**.
- **sem_nsems** is set to the value of **nsems.**
- **sem_otime** is set to zero, and **sem_ctime** is set to the current AIX operating system time.

semget fails if one or more of the following are true:

[EINVAL] **nsems** is either less than or equal to zero or greater than the AIX operating system-imposed limit.

[EACCES] A semaphore identifier exists for **key**, but operation permission (see **intro**) as specified by the low-order nine bits of **semflg** would not be granted.

[EINVAL] A semaphore identifier exists for **key**, but the number of semaphores in the set associated with it is less than **nsems** and **nsems** is not equal to zero.

[ENOENT] A semaphore identifier does not exist for **key**, and (**semflg & IPC_CREAT**) is "false."

[ENOSPC] A semaphore identifier is to be created, but the number of semaphore identifiers would exceed the system maximum allowed for that host.

[EEXIST] A semaphore identifier exists for **key**, but (**semflg & IPC_CREAT**) & (**semflg & IPC_EXCL**) is "true."

RETURN VALUES

On successful completion, a semaphore identifier returns. Otherwise, a value of −1 returns and **errno** is set to indicate the error.

SEE ALSO

dflthost
intro
semctl
semop

NAME

semop – semaphore operations

SYNOPSIS

```
#include <pcilib.h>
#include <memmdl.h>
#include <ipc.h>

int semop (semid, sops, nsops)
int semid;
struct sembuf **sops;
int nsops;
```

DESCRIPTION

semop is used to perform an array of semaphore operations on the set of semaphores associated with the semaphore identifier specified by **semid**. The operations are performed on the current host (as specified in the most recent **dflthost** call). **sops** is a pointer to the array of semaphore-operation structures. **nsops** is the number of such structures in the array. Each structure includes the following members:

```
short   sem_num;    /* semaphore number */
short   sem_op;     /* semaphore operation */
short   sem_flg;    /* operation flags */
```

Each semaphore operation specified by **sem_op** is performed on the corresponding semaphore specified by **semid** and **sem_num**.

sem_op specifies one of three semaphore operations as follows:

1. If **sem_op** is a negative integer, one of the following occurs:

   If **semval** (see **intro**) is greater than or equal to the absolute value of **sem_op**, the absolute value of **sem_op** is subtracted from **semval**. Also, if (**sem_flg & SEM_UNDO**) is "true," the absolute value of **sem_op** is added to the calling process's **semadj** value for the specified semaphore. (See **exit** in the *AIX Operating System Technical Reference.*)

   If **semval** is less than the absolute value of **sem_op** and (**sem_flg & IPC_NOWAIT**) is "true," **semop** returns immediately.

   If **semval** is less than the absolute value of **sem_op** and (**sem_flg & IPC_NOWAIT**) is "false," **semop** returns −1 and **errno** is set to EIDRM.

2. If **sem_op** is a positive integer, the value of **sem_op** is added to **semval**, and if **(sem_flg & SEM_UNDO)** is "true," the value of **sem_op** is subtracted from the calling process's **semadj** value for the specified semaphore.

3. If **sem_op** is zero, one of the following occurs:

   If **semval** is zero, **semop** returns immediately.

   If **semval** is not equal to zero and **(sem_flg & IPC_NOWAIT)** is "true," **semop** returns immediately.

   If **semval** is not equal to zero and **(sem_flg & IPC_NOWAIT)** is "false," **semop** returns −1 and **errno** is set to EIDRM.

semop fails if one or more of the following are true for any of the semaphore operations specified by **sops**:

[**EINVAL**] **semid** is not a valid semaphore identifier.

[**EFBIG**] **sem_num** is less than zero or greater than or equal to the number of semaphores in the set associated with **semid**.

[**E2BIG**] **nsops** is greater than the AIX operating system-imposed maximum.

[**EACCES**] Operation permission is denied to the calling process (see **intro**).

[**EAGAIN**] The operation would result in suspension of the calling process, but **(sem_flg & IPC_NOWAIT)** is "true."

[**ENOSPC**] The AIX operating system limit on the number of individual processes requesting a **SEM_UNDO** would be exceeded.

[**EINVAL**] The number of individual semaphores for which the calling process requests a **SEM_UNDO** would exceed the AIX operating system limit.

[**ERANGE**] An operation would cause a **semval** to overflow the AIX operating system-imposed limit.

[**ERANGE**] An operation would cause a **semadj** value to overflow the AIX operating system-imposed limit.

[**EFAULT**] **sops** points to an illegal address.

On successful completion, the value of **semid** for each semaphore specified in the array pointed to by **sops** is set to the process ID of the DOS server.

RETURN VALUES

On successful completion, the value of **semval** at the time of the call for the last operation in the array pointed to by **sops** returns.

If **semop** returns due to the removal of a **semid** from the system, a value of −1 returns and **errno** is set to EIDRM.

Otherwise, a value of −1 returns and **errno** is set to indicate the error.

SEE ALSO

dflthost
intro
semctl
semget
sleep_h

BUGS

Any requested **UNDO** operations will occur on a particular host when you log out of that host, rather than when the DOS application exits.

DIFFERENCE FROM SYSTEM V

Only nonblocking operations are allowed from DOS programs. If a condition arises that would cause the DOS process to block, the function returns an error. Use **sleep_h** to suspend a DOS program.

NAME

sleep_h – suspend DOS execution

SYNOPSIS

```
void sleep_h(h)
unsigned short h;
```

DESCRIPTION

sleep_h() suspends the calling process for approximately *h* hundredths of a second. Interrupt routines continue to execute as interrupts occur during the sleep.

In a DOS Merge environment, **sleep_h()** actually relinquishes the processor to other AIX operating system processes, waking up and checking on each virtual interrupt to see whether the requested amount of time has elapsed. When the requested time has elapsed, **sleep_h()** returns to the caller.

The use of **sleep_h()** is recommended in preference to busy-wait loops, which degrade overall system performance when run in a virtual DOS environment on a timeshare system.

IMPLEMENTATION NOTES

The requested duration of sleep time is rounded to the nearest DOS virtual tick, where one DOS virtual tick interval is approximately 5.5 hundredths of a second. (More precisely, DOS virtual ticks occur at the rate of (1193180/65536) per second; **sleep_h()** assumes a conversion to the rate of (1193200/65536) DOS virtual ticks per second.)

sleep_h() waits for the appropriate number of DOS virtual ticks to occur, and then returns to the caller. Note that the amount of time that actually passes before the first tick occurs depends on how much time has already passed in the current tick interval when **sleep_h()** is called.

If the requested duration of sleep time rounds to zero DOS virtual ticks, the calling process is suspended until the next interrupt. (This will not necessarily be a clock interrupt.)

If the calling process is running in a virtual DOS environment under DOS Merge and **fastclk** is not set, DOS virtual time is only updated approximately once every second, or once every 18 DOS virtual ticks. Thus, for delays on the order of several seconds or more, the time slept will be reasonably accurate, but for smaller delays the granularity may be unacceptable.

If the **fastclk** option is enabled under DOS Merge, then DOS virtual time is updated at every DOS virtual tick interval, just as it is in a standard DOS environment.

Even if **fastclk** is set, AIX operating system process scheduling will affect the actual time slept in a virtual DOS environment.

If the DOS clock has been reprogrammed, then **sleep_h()** should not be used since it assumes the default clock rate.

EXAMPLE

```
        :
        :
int  driveNum;
int  status;
int  blockFlag;
pid_t retVal;
        :
        :
/*
** Use sleep_h() with all nonblocking functions.
** In this example we use the sleep_h() with the
** uwait() function. (see MAN page for uwait()).
**
** Use UWAIT to wait for a child process
** on virtual drive D: (4) to exit.
*/
driveNum = 4;
blockFlag = 0;    /* nonblocking */

/* loop till a child exits */
while ( (retVal = uwait(driveNum, &status, blockFlag)) == 0L )
        sleep_h(25);       /* sleep for a quarter of a second */

if (retVal == -1L)
        errorHandler();
else
        checkStatus();

        :
        :
```

SEE ALSO

The command **merge set fastclk [on|off]** in the DOS Merge documentation.

NAME

uchmod – change AIX operating system file attributes

SYNOPSIS

```
#include <pcilib.h>
#include <memmdl.h>

int uchmod( dpath, mode )
char *dpath;
int mode;
```

DESCRIPTION

uchmod is a DOS function used to change the file attribute or mode of a file on an AIX operating system.

mode is the target AIX operating system file mode. Refer to **chmod** in the *AIX Operating System Technical Reference Manual* for a description of the various bits in the **mode** parameter.

dpath is a DOS style path name for a file residing on the virtual drive whose mode is to be changed. The path may include a drive designation to indicate on which host the operation is to be performed. If no drive is specified, the current drive is used.

RETURN VALUES

On successful completion, a value of 0 returns. If the call fails for any reason, a value of –1 returns and a DOS error code is returned in **errno**.

EXAMPLE

```
        :
        :
char *dPath = "D:\\EXISTING\\FILENAME";
int  mode   = 0755;        /* rwxr-xr-x permission in octal */
int  retVal;
        :
        :
/* Change the mode of the DOS file name on virtual drive D: */
if ( (retVal = uchmod(dPath, mode)) == -1 )
        errorHandler();
        :
        :
```

SEE ALSO

getuattr
isvirtual

NAME

uexec – execute AIX operating system command

SYNOPSIS

```
#include <pcilib.h>
#include <memmdl.h>

long uexec(drvnum, ucmdname, argv, envp,
           savestatus, error_code)
int drvnum;
char *ucmdname, *argv[], *envp[];
int savestatus, *error_code;
```

DESCRIPTION

uexec is a DOS function call that starts the execution of an AIX operating system command on a host and returns the process ID. The AIX operating system **stdout**, **stderr**, and **stdin** are all directed to **/dev/null**.

drvnum is the drive number of the host on which the command is executed. Drive letters and numbers correspond in the following way: A: = 1, B: = 2, and so forth. If **drvnum** is 0, the program is executed on the host associated with the current drive.

ucmdname points to a path name that identifies the AIX operating system and the file to be executed. This can be a load module or a shell command file. If the name contains no slash (/), the directories in the PATH environment variable of the new process are searched in order.

argv is an array of character pointers to the arguments for the command. By convention, **argv[0]** points to the command name. **argv** is terminated by a null pointer.

envp is an array of character pointers to null-terminated strings. These strings are combined with the environment strings of the server process to constitute the environment for the new process. If one of the environment strings in **envp** defines an environment variable with the same name as an environment variable already in the server, the string in **envp** overrides the string inherited by the server. **envp** is terminated by a null pointer.

savestatus is an integer taking the following values:

1 Save exit status on termination.
0 Do not save exit status on termination.

When the process exits, the exit status is optionally saved depending on the value of the **savestatus** flag. The DOS **uwait** function may be used to poll for the exit status of the process.

RETURN VALUES

On successful completion, the process ID of the child process returns. If the call fails for any reason, a value of −1 returns, and **error_code** is set to indicate the error.

The error codes for this call are:

1 Error in request service (network error).
2 The specified **drvnum** is not connected to a host.
3 AIX operating system **exec** failed.
4 Invalid format.
5 DOS memory allocation error.

If the AIX operating system file server process exits because the user logs out, any AIX operating system processes started with **uexec** are killed unless they take steps to avoid this (see **signal** in the *AIX Operating System Technical Reference*).

When using shell features such as I/O redirection, use **sh** or **csh** as the **cmdname** and **-c** as the first argument. The command line being executed by the shell should be the second argument.

Note: The return value is a 32-bit integer.

EXAMPLE

```
        :
        :
int  driveNum;
long pid;
int  saveStatus;
int  errorCode;
        :
        :
/*
** AIX command to execute:
**     who | sort > /tmp/result
**
** Since our command has shell features, (i.e. "|" and ">")
** we need to UEXEC the Shell ("/bin/sh") and then pass
** our command as an argument to it.
** If we had a simple AIX command which had no shell features,
** (for example "touch  filename") we would have simply UEXEC'ed
** the command itself, and there would have been no need for shell.
*/
```

```
/* AIX command being UEXEC'ed */
char *uCommand = "/bin/sh";

/* Create UEXEC argument string (i.e.: sh -c args) */
char *args[] = {
       "sh",
       "-c",
       "who | sort > /tmp/result",
       NULL
       };

/* Set environment */
char *envs[] = {
       "PATH=.:/bin:/usr/bin:/usr/lbin:",
       "TZ=PST8PDT",
       NULL
       };

       :
       :
/*
** Execute the AIX command on virtual drive
** D: (4), saving its exit status (saveStatus=1).
*/
driveNum = 4;
saveStatus = 1;
if ((pid = uexec(driveNum,uCommand,args,envs,saveStatus,&errorCode))
       == -1L)
       errorHandler();
else

       :
       :
```

SEE ALSO

ukill
uwait

NAME

ukill – send a signal to an AIX operating system process or group of processes

SYNOPSIS

```
#include <pcilib.h>
#include <memmdl.h>
#include <sys/types.h>
#include <signal.h>

int ukill( drvnum, pid, sig )
int drvnum;
pid_t pid;
int sig;
```

DESCRIPTION

ukill sends a signal to a process or a group of processes. **pid** specifies the process or group of processes to which the signal is to be sent. **drvnum** is the drive number of the virtual drive connected to the host running the process to which the signal is to be sent. (Drive A: = 1, drive B: = 2, and so forth.) If **drvnum** is 0, the signal is sent on the host associated with the current drive. The signal to be sent is specified by **sig** and is either one from the list given in the *AIX Operating System Technical Reference* for **signal** or 0. For compatibility with 4.1 BSD programs, **sig** may also be the negative of one of the listed signal numbers. If **sig** is 0 (the null signal), error checking is performed but no signal is actually sent. This can be used to check the validity of **pid**.

The real or effective user ID of the sending DOS server process must match the real or effective user ID of the receiving process, unless the effective user ID of the DOS server process is super-user.

Several processes, including those with process IDs of 0 and 1, are special system-created processes. Of these, only process 1 (the **initialization process**) may ever be signaled.

If **pid** is greater than 0 and **sig** is greater than or equal to 0, **sig** will be sent to the process whose process ID is equal to **pid**. **pid** may equal 1, but may not be the process ID of any other system-created process.

If **pid** is 0, the absolute value of **sig** will be sent to each process (excluding the system-created processes) whose process group ID is equal to the process group ID of the sender. The sender's process group ID may not be 0.

If **pid** is −1 and the effective user ID of the sender is not super-user, the absolute value of **sig** will be sent to each process whose real user ID is equal to the effective user ID of the sender.

If **pid** is −1 and the effective user ID of the sender is super-user, the absolute value of **sig** will be sent to all processes excluding the system-created processes.

If **pid** is negative but not −1, the absolute value of **sig** will be sent to each process whose process group ID is equal to the absolute value of **pid**. This is the standard AIX method for sending signals to members of a specified process group.

If **pid** is positive and **sig** is negative, the absolute value of **sig** will be sent to each process whose process group ID is equal to **pid**. This is the standard 4.1 BSD method for sending signals to members of a specified process group.

ukill fails and no signal is sent if one or more of the following are true:

[EINVAL] **sig** is not a valid signal number.

[EINVAL] **pid** is the ID of one of the system-created processes that cannot be signaled.

[ESRCH] No process can be found corresponding to that specified by **pid**.

[ESRCH] **pid** is 0 and the sender's process group ID is 0.

[EPERM] The user ID of the sending process is not super-user, and its real or effective user ID does not match the real or effective user ID of the receiving process.

RETURN VALUES

On successful completion, a value of 0 returns. If the call fails for any reason, a value of −1 returns, and **errno** is set to indicate the error.

EXAMPLE

```
                :
                :
#define SIGTERM  15     /* See AIX  signal(2) man page */

int     driveNum;
pid_t   pid;
int     sig;
int     retVal;
                :
                :
/*
** Send Termination signal (SIGTERM),
** to a process with PID 7476 on virtual
** drive D: (4).
*/
sig = SIGTERM;
pid = 7476;
driveNum = 4;

if ( (retVal = ukill(driveNum, pid, sig)) == -1 )
                errorHandler();
else

                :
                :
```

SEE ALSO

uexec
uwait

NAME

uren – rename an AIX operating system file

SYNOPSIS

```
#include <pcilib.h>
#include <memmdl.h>

int uren( oldupath, newupath )
char *oldupath, *newupath;
```

DESCRIPTION

uren is a DOS function used to rename AIX operating system files. The file to be renamed must exist, and the DOS user must have access to it.

oldupath is the AIX operating system name of the file to be renamed. The name may include a drive designation to select the host. If no drive is specified, the current drive is used.

newupath is the path name to be given to the file.

RETURN VALUES

If any error occurs, a value of –1 returns.

EXAMPLE

```
        :
        :
char *oldUpath = "D:/aixpath/file1";
char *newUpath = "D:/aixpath/file2";
int  retVal;
        :
        :
/*
** Rename the AIX file on
** virtual drive D:
*/
if ( (retVal = uren(oldUpath, newUpath)) == -1 )
        errorHandler();

        :
        :
```

SEE ALSO

isvirtual
mapd2u

NAME

uwait – poll for the exit status of an AIX operating system process

SYNOPSIS

```
#include <pcilib.h>
#include <memmdl.h>

pid_t uwait(drvnum, status, blockflag)
int drvnum, *status, blockflag;
```

DESCRIPTION

uwait is a DOS function call that checks to see if an AIX operating system command started with the **uexec** function has terminated. The AIX operating system program must have been executed with the option to save the exit status.

drvnum is the drive number of the host on which to check. Drive letters and numbers correspond in the following way: A: = 1, B: = 2, and so forth. If **drvnum** is 0, the **uwait** is performed on the host associated with the current drive.

The **blockflag** controls whether the call is blocking or nonblocking. If **blockflag** is "true", the call blocks. If **blockflag** is "false", the call does not block. Only the nonblocking form of this call is now supported. Use **sleep_h** to suspend a DOS program.

RETURN VALUES

The return values for this call are:

-1 No children for which status is being saved.
0 No children have exited.
>0 Process ID of exited child and status is set.

The low byte of **status** is the exit status supplied by the terminating process or, if it was terminated by a signal, the signal that terminated it. The following values return in the high byte of the status field:

0 Low byte is exit status.
1 Low byte is signal terminating the process.
2 Low byte is signal terminating process and generating a core dump.
3 Process was not started by **uexec** or status already collected.

EXAMPLE

```
        :
        :
int  driveNum;
int  status;
int  blockFlag;
pid_t retVal;
        :
        :
/*
** Use UWAIT to wait for a child process
** on virtual drive D: (4) to exit.
**
** Since only the nonblocking form of this
** function is supported (blockFlag = 0), we
** use PCILIB's sleep_h() function in our
** busy loop.
*/
driveNum = 4;
blockFlag = 0;

/* loop till a child exits */
while ( (retVal = uwait(driveNum, &status, blockFlag)) == 0L )
        sleep_h(25);        /* sleep for quarter of a second */

if (retVal == -1L)
        errorHandler();
else
        checkStatus();

        :
        :
```

SEE ALSO

sleep_h
uexec
ukill

NAME

vdrive – return the drive number of a virtual drive

SYNOPSIS

```
#include <pcilib.h>
#include <memmdl.h>

int vdrive( vdrvnum )
int vdrvnum;
```

DESCRIPTION

vdrive is a function that returns the DOS drive number of a virtual drive.

vdrive(0) returns the total number of virtual drives.

vdrive(vdrvnum) returns the DOS drive number of virtual drive **vdrvnum**. **vdrvnum** must be a value between 1 and **vdrive(0)**.

RETURN VALUES

The drive numbers returned by **vdrive(*n*)** are 1 for drive A:, 2 for drive B:, and so on. These drives may or may not be currently logged in.

If no virtual drives are associated with the personal computer or the **vdrvnum** argument is greater than **vdrive(0)**, a value of 0 returns.

EXAMPLE

```
        :
        :
int driveNum;
int drive;
int i;
        :
        :
/* First get the number of virtual drives.*/
if ( (driveNum = vdrive(0)) == 0 ) {
        /* no virtual drives */
        errorHandler();
} else {
        printf("Number of virtual drives: %d \n", driveNum);
        /* Get DOS drive letter for each virtual drive.*/
        for (i = 1; i <= driveNum; i++) {
             drive = vdrive(i);
             printf("Virtual drive %d = DOS drive %c:\n",i,
                    drive+'A'-1);
}
        :
        :
```

SEE ALSO

isvirtual
vhost

NAME

vhost – return the name of a currently connected host

SYNOPSIS

```
#include  <pcilib.h>
#include  <memmdl.h>

char* vhost( drvnum )
int drvnum;
```

DESCRIPTION

vhost is a DOS function call that returns the name of the host attached to the virtual drive specified by **drvnum.**

drvnum is the drive number of the host whose name is being requested. Drive letters and numbers correspond in the following way: A: = 1, B: = 2, and so forth.

RETURN VALUES

If **drvnum** corresponds to a logged-in virtual drive, **vhost** returns the name of the host attached to the virtual drive specified by **drvnum.**

If **drvnum** is out of range, or the drive that was selected is currently not connected to a host (not logged in), then **vhost** returns NULL.

EXAMPLE

```
        :
        :
char hostName[15];
char *hostPtr;
int  driveNum;
        :
        :
/* Get the Host name for virtual drive D: (driveNum = 4) */
driveNum = 4;
if ( (hostPtr = vhost(driveNum)) == NULL )
        errorHandler();
else {
        /* Save a copy of host name */
        strcpy(hostName, hostPtr);
        printf("Host name for Drive %c: is %s", driveNum+'A'-1, hostName);
}
        :
        :
```

SEE ALSO

isvirtual
vdrive

BUGS

The name returned is stored in a static buffer. Subsequent calls to **vhost** overwrites this buffer.

IBM AIX Family

Reader's Comment Form

PCILIB, AIX Access for DOS Users Extended Library SC23–2360–00

Your comments assist us in improving our products. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

For prompt resolution to questions regarding set up, operation, program support, and new program literature, contact your IBM representative, your authorized IBM dealer, or authorized IBM remarketer in your area.

Comments:

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 997, Building 997
11400 Burnet Rd.
Austin, Texas 78758–3493

Fold

Fold

Cut or Fold Along Line

Fold and Tape

Please Do Not Staple

Fold and Tape

©IBM Corp. 1987,1991
All rights reserved

International Business
Machines Corporation
11400 Burnet Road
Austin, Texas 78758

Printed in the
United States of America

SC23–2360–00